A Constant-Factor Approximation Algorithm for the Multicommodity Rent-or-Buy Problem

Amit Kumar [*] Anupam Gupta [†] Tim Roughgarden [‡]

Abstract

We present the first constant-factor approximation algorithm for network design with multiple commodities and economies of scale. We consider the rent-or-buy *problem, a type of multicommodity buy-at-bulk network design in which there are two ways to install capacity on any given edge. Capacity can be* rented, *with cost incurred on a per-unit of capacity basis, or* bought, *which allows unlimited use after payment of a large fixed cost. Given a graph and a set of source-sink pairs, we seek a minimum-cost way of installing sufficient capacity on edges so that a prescribed amount of flow can be sent simultaneously from each source to the corresponding sink.*

Recent work on buy-at-bulk network design has concentrated on the special case where all sinks are identical; existing constant-factor approximation algorithms for this special case make crucial use of the assumption that all commodities ship flow to the same sink vertex and do not obviously extend to the multicommodity rent-or-buy problem. Prior to our work, the best heuristics for the multicommodity rent-or-buy problem achieved only logarithmic performance guarantees and relied on the machinery of relaxed metrical task systems or of metric embeddings. By contrast, we solve the network design problem directly via a novel primal-dual algorithm.

1 Introduction

We consider the problem of network design with multiple commodities and economies of scale. More precisely, given an undirected graph $G = (V, E)$ and a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_p, t_p)\}$ of vertex pairs called *demands*, we seek a minimum-cost way of installing sufficient capacity on the edges E so that a prescribed amount of flow can be sent simultaneously from each source s_k to the corresponding sink t_k. We are interested in the scenario where the cost of installing capacity exhibits economies of scale, in the sense that buying a large amount of capacity on a single edge results in a high capacity-to-cost ratio (i.e., good "bang for your buck"). Put differently, the cost of capacity is a concave function of the capacity bought.

The general problem described above goes by the name *buy-at-bulk network design*, and was introduced by Salman et al. [23]. The problem is NP-hard [23], and researchers have therefore sought out good approximation algorithms for the problem. The best algorithm currently known for the general problem is due to Awerbuch and Azar [3], who give an $O(\log n \log \log n)$-approximation based on Bartal's method for probabilistically embedding general metrics into tree metrics [5], where n is the number of nodes in the network. Improvements on the algorithm of [3] have been elusive, leading researchers to consider special cases of the problem. The *single-sink* version of buy-at-bulk network design, where all sinks t_k are identical, has recently received much attention. Andrews and Zhang [2] designed an $O(K^2)$-approximation algorithm for the single-sink problem when the cost of installing capacity is a restricted type of concave piecewise linear function with K breakpoints. This problem is called *access network design* in [2]. A constant-factor approximation algorithm for the access network design problem was later given by Guha et al. [12]. Subsequently and independently, Garg et al. [7] gave an $O(K)$-approximation algorithm and Guha et al. [13] designed a constant-factor approximation algorithm for the general single-sink buy-at-bulk network design problem (with an arbitrary concave, piecewise linear function describing the cost of installing a given amount of capacity). The constant of [13] has recently been improved upon by Talwar [25].

Despite these recent successes for the single-sink problem, there have been few improvements over the algorithm of Awerbuch and Azar [3] for any nontrivial version of *mul-*

[*] Lucent Bell Labs, 600 Mountain Avenue, Murray Hill NJ 07974. This work was done while the author was at Cornell University, Ithaca NY 14853, and supported by NSF ITR/IM Grant IIS-0081334 and the ONR Young Investigator Award of Jon Kleinberg. Email: `amitk@research.bell-labs.com`.

[†] Lucent Bell Labs, 600 Mountain Avenue, Murray Hill NJ 07974. Email: `anupamg@research.bell-labs.com`.

[‡] Department of Computer Science, Cornell University, Ithaca NY 14853. Supported by an NSF Graduate Fellowship, a Cornell University Fellowship, and ONR grant N00014-98-1-0589. Email: `timr@cs.cornell.edu`.

ticommodity buy-at-bulk network design. In this paper, we present the first constant-factor approximation algorithm for such a problem. As all of the techniques employed in recent papers [2, 7, 12, 13, 17, 25] make crucial use of the assumption that all commodities ship flow to the same sink vertex and do not obviously extend to the multicommodity setting, our algorithm and analysis require several new ideas. We also avoid reliance on metric embedding techniques (unlike [3]), instead attacking the network design problem directly via a novel primal-dual algorithm.

The Rent-or-Buy Problem. In this paper, we consider the *rent-or-buy* problem, a type of multicommodity buy-at-bulk network design with a simple function describing the cost of installing capacity. In the rent-or-buy problem, there are two ways to install capacity on any given edge. Capacity can be *rented*, with cost incurred on a per-unit of capacity basis, or *bought*, which allows unlimited use after payment of a large fixed cost. We model this scenario with positive parameters μ and M, with the cost of renting capacity equal to μ times the capacity required (per unit length), and the cost of buying capacity equal to M (per unit length). There is no loss of generality in assuming that $\mu = 1$. The multicommodity rent-or-buy problem was previously studied in an online setting by Awerbuch et al. [4] (where it was called the *network connectivity leasing problem*) and Bartal et al. [6], who used the framework of relaxed metrical task systems to give $O(\log^2 n)$- and $O(\log n)$-competitive algorithms for the problem, respectively.

Buy-at-bulk network design was originally defined in terms of installing *cables* on edges, with different cable types offering different amounts of capacity and carrying different costs [3, 23]. Andrews and Zhang [2] showed that this problem can be rephrased (with a loss of a small constant factor in the approximation ratio) with each cable type carrying a *fixed cost* (which must be paid irrespective of the capacity needed) and an *incremental cost* (which is paid for each unit of capacity required). The rent-or-buy problem therefore corresponds to the special case of one cable type with an incremental cost but no fixed cost, and one cable type with a fixed cost but no incremental cost.

We believe the rent-or-buy problem captures much of the essence of buy-at-bulk network design. Most of the difficulty of network design problems in which capacities obey economies of scale stems from the following tension: on the one hand, we would like to route flow between a source and sink on an (approximately) shortest path; on the other, we would like to gather flow from many different commodities together in order to purchase large quantities of capacity and take advantage of economies of scale. This issue of "route vs. gather" is clearly present in the rent-or-buy problem, and we believe that overcoming the difficulties caused by multiple commodities in this simple setting will lead to further progress on the general multicommodity buy-at-bulk network design problem.

Application to Maybecast. In addition to being a nontrivial special case of buy-at-bulk network design, the rent-or-buy problem arises in important applications. For example, Karger and Minkoff [17] introduced the so-called *maybecast* problem, defined as follows. There is an underlying undirected network G, with a source vertex s from which a multicast transmission will emanate, and a set D of demand vertices that wish to receive the transmission. The problem of building the min-cost network that connects the source to all of the demands is the classical min-cost Steiner tree problem. Karger and Minkoff [17] proposed a probabilistic version of this problem: each demand vertex i contacts the source s independently with probability p_i. Relative to a fixed Steiner tree on $\{s\} \cup D$, when a demand contacts the source s, all edges on the path joining it to s are said to become *active*. The goal is then to build the Steiner tree that minimizes the expected cost of the active edges.

Our solution to the rent-or-buy problem provides a constant factor approximation for the following *multicommodity* version of the maybecast problem. Instead of a single source s, we are given a set of sources S. Each demand wants to receive data from one source in S, and it contacts that source with some probability. As in the previous problem, we seek paths between the demands and the sources they wish to contact so that the expected number of active edges is minimized. This problem reduces, modulo a small constant factor in the approximation ratio, to rent-or-buy network design (see [17]).

Application to Connected Facility Location. Our results also give a constant-factor approximation algorithm for a multicommodity version of *connected facility location*, a problem that has recently received attention in both the operations research literature [19, 20, 21] and the computer science community [14, 17, 18]. In the previously studied version of the connected facility location problem, the input is a set F of *facilities*, a set D of *demands*, a graph $G = (V, E)$ with $V = F \cup D$ and costs c_e on edges e, and a parameter $M > 1$. A solution consists of an assignment of demands to facilities and a subgraph of G spanning the open facilities (a Steiner tree). If demand j is assigned to facility $i(j)$ and the length of the shortest path between them in G (w.r.t. c) is $d(j, i(j))$, then the cost of a solution is $\sum_{j \in D} d(j, i(j)) + M \sum_{e \in T} c_e$ (where T is the Steiner tree spanning the open facilities). The first constant-factor approximation algorithm for this problem was given by Karger and Minkoff [17], and Gupta et al. [14] subsequently gave an algorithm with improved performance guarantee. Very recently, Swamy and Kumar [24] obtained a 5-approximation algorithm for this problem.

In the multicommodity version of connected facility location, we are in addition given several *commodities*. Each demand belongs to one of these commodities. We again open facilities and assign demands to them, but now require only a subgraph T such that, for any commodity k, the set of facilities serving demands of commodity k are connected. In solving the rent-or-buy problem, we develop techniques that also give a constant-factor performance guarantee for the multicommodity connected facility location problem.

New Techniques for Primal-Dual Approximation Algorithms. Our algorithm is based on the primal-dual method. The high-level idea of this method is to consider an integer programming formulation of our network design problem and the dual of its linear programming relaxation, and to iteratively construct both an integral primal solution (i.e., a feasible network) and a feasible dual solution proving that the network has near-optimal cost.

The first systematic application of the primal-dual method was to a large class of network design problems; see [11, 26] for a survey of this and earlier work. More recently, Jain and Vazirani [16] gave primal-dual approximation algorithms for several facility location problems that could not be solved using earlier techniques. Our algorithm is at times reminiscent to the facility location algorithms of [16] (reflecting our need to cluster demands together to leverage economies of scale) and to the network design algorithms described in [11] (as clustered demands must then be connected cheaply, as in canonical network design problems). However, these two implementations of the primal-dual method are not easily combined, and we require further ideas to obtain a good approximation algorithm for the rent-or-buy problem. In particular, we contribute two new techniques to existing primal-dual technology that we believe may find other applications.

First, we introduce *geometric scaling* in a primal-dual context. We use scaling to break up the execution of our algorithm into successive stages in a way that ensures that the "mistakes" made in any given stage have little significance for future stages. While other primal-dual algorithms have been used as a black-box within a scaling procedure [1, 8, 9, 27], we use scaling *inside* our primal-dual algorithm to control the rate of increase of dual variables.

Second, unlike most previous primal-dual approximation algorithms, we do not explicitly maintain feasibility of our dual solution. Rather, we maintain feasibility with respect to a strict subset of the dual constraints, and prove that we are always *approximately* feasible for the full LP. This idea is similar in spirit to recent "dual fitting" approaches to facility location problems [15, 22]. Freed from the need to maintain dual feasibility, we can make use of an unusually aggressive dual increase step; this in turn allows us to more easily argue that the cost of our solution is close to the objective function value of our (approximately feasible) dual solution.

2 Preliminaries

An instance of multicommodity rent-or-buy network design (MROB) is specified by an undirected graph $G = (V, E)$, a nonnegative cost c_e for each edge e, a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_p, t_p)\}$ of pairs of demands, and a parameter $M > 1$. We will abuse notation and write $j \in \mathcal{D}$ for a generic demand j of the form s_k or t_k. We assume for simplicity that the source s_k wishes to send one unit of flow to the sink t_k, but our algorithm and analysis extend without difficulty to non-uniform flow requirements (details omitted from this abstract). By $d(u, v)$ we mean the length of the shortest path[1] in G between vertices u and v, with respect to edge lengths c.

A solution to an MROB instance is specified by an assignment of each demand pair $(s_k, t_k) \in \mathcal{D}$ to an s_k-t_k path of G. If a_e paths use edge e, then the cost of this solution is defined by $\sum_{e \in E} c_e \min\{a_e, M\}$. The term $c_e a_e$ corresponds to renting capacity on edge e, and the term $c_e M$ corresponds to buying capacity on e. We seek a solution of minimum cost.

2.1 Reformulation as Connected Facility Location

We begin with a reduction from MROB to multicommodity connected facility location (MCFL). We will see shortly that the latter problem admits a relatively simple integer programming formulation, thereby allowing us to use the primal-dual method.

The precise problem that we reduce to is the following. The input is an undirected graph $G = (V, E)$ with edge e possessing cost c_e, a set $\mathcal{D} = \{(s_1, t_1), \ldots, (s_p, t_p)\}$ of vertex pairs, and a parameter $M > 1$. A solution consists of a set $F \subseteq V$ of facilities to open, an assignment of sources and sinks to open facilities, and a subgraph (V, H) of G with the following property: if for some k, s_k is assigned to facility i_1 and t_k to i_2, then there is a path in (V, H) between i_1 and i_2. The cost of a solution is $\sum_{j \in \mathcal{D}} d(j, i(j)) + M \sum_{e \in H} c_e$, where $i(j)$ is the facility to which the demand j is assigned and d is again shortest-path distance in G (with respect to c). (The seemingly more general statement of MCFL in Section 1 can also be reduced to the one above.) We then have the following reduction.

Lemma 2.1 *A β-approximation algorithm for MCFL gives a 2β-approximation algorithm for MROB.*

Proof: An instance of MROB naturally defines an instance of MCFL with the same parameters (G, c, $\mathcal{D}$, and M). We will map every solution of the latter problem to one of the former with equal cost, and an optimal solution to the former problem to one of the latter with at most twice the cost.

[1]Throughout this paper, we assume some arbitrary but fixed tie-breaking mechanism that ensures uniqueness of shortest paths.

Given a solution to the induced MCFL instance, define an MROB solution as follows. The s_k-t_k path is defined to be the shortest path from s_k to $i(s_k)$ and from $i(t_k)$ to t_k, connected by a path from $i(s_k)$ to $i(t_k)$ in H, the subgraph of edges chosen in the facility location solution (which exists by feasibility for MCFL). This solution to the MROB instance has cost bounded above by the MCFL solution.

Consider an optimal solution $P_1^*, \ldots, P_p^*$ to an MROB instance. Let H^* denote the edges used by M or more paths. We cannot simply reverse the mapping of the previous paragraph, since there is no guarantee that the subpaths of P_k^* from s_k to H^* and from t_k to H^* terminate in a common component of H^*. Instead, initialize H (the connecting edges in our facility location solution) to be H^* and F (the open facilities) to be the vertices spanned by H^*; we will supplement these sets with further edges and vertices shortly. Define $\{H_i\}$ to be the components of (V, H) with isolated vertices discarded; this set is initially just the nontrivial components of (V, H^*), but will change as we add further edges to H.

Call a demand pair (s_k, t_k) *good* if path P_k^* is edge-disjoint from all but at most one H_i, and *bad* otherwise. If P_k^* is edge-disjoint from all H_i's, then add vertex s_k to F and assign both s_k and t_k to it. If P_k^* intersects only H_i, then assign each of s_k and t_k to their nearest neighbors in H_i. As long as there is a bad pair, we execute the following procedure.

Let H_i be the component of minimum index that intersects some bad demand demand pair, say (s_k, t_k). Call H_i the *current component*. Let P_k^1 denote the edges of $P_k^* \setminus H$, and P_k^2 the edges of $P_k^* \cap H$ that lie outside of H_i (in components with larger index). Our analysis breaks into two cases. Let $c(P)$ denote $\sum_{e \in P} c_e$ for a subgraph P.

Case 1: Suppose $c(P_k^1) \geq c(P_k^2)$. In this case we assign each of s_k and t_k their nearest neighbors in H_i, and redefine the demand pair (s_k, t_k) to be good.

Case 2: Suppose $c(P_k^1) < c(P_k^2)$. In this case we add all edges of P_k^1 to H, and add all endpoints of these edges to F. Since (s_k, t_k) is bad, this addition causes two or more components (H_i and components of higher index) to merge into a single component; the new component retains the index i. Any demand pair (s_q, t_q) whose path is now edge-disjoint from all non-trivial components of (V, H) except H_i (such as (s_k, t_k)) is redefined to be good, and s_q and t_q are assigned to their nearest neighbors in H_i.

Each iteration of the above procedure strictly increases the number of good demand pairs and maintains the invariant that all good demand pairs have been assigned to open facilities in a common component of (V, H). The procedure therefore terminates with a feasible solution to the MCFL instance; it remains to show that this solution has small cost.

We first claim that assignment costs of our solution are at most $2 \sum_{e \notin H^*} a_e c_e$, where $a_e < M$ paths of the network design solution use edge e. It suffices to show that, for each demand pair (s_k, t_k), our assignment costs for s_k and t_k are upper bounded by twice the cost of the edges in $P_k^* \setminus H^*$. This is clear for a demand pair (s_k, t_k) whose path P_k^* is edge-disjoint from H^*, since its assignment cost is $d(s_k, t_k) \leq c(P_k^*) = c(P_k^* \setminus H^*)$. Suppose now that at some point in the procedure, the demand pair (s_k, t_k) got assigned because its path P_k^* intersected the current graph (V, H) in exactly one component, say H_i. Since s_k and t_k are assigned to their nearest neighbors in H_i, it is then easy to see that $d(s_k, i(s_k)) + d(t_k, i(t_k)) \leq c(P_k^* \setminus H_i) \leq c(P_k^* \setminus H^*)$. Finally, suppose demand pair (s_k, t_k) is assigned in case 1 of some iteration of the procedure, with H_i the current component. Since s_k and t_k are assigned to nearest neighbors in H_i, we have $d(s_k, i(s_k)) + d(t_k, i(t_k)) \leq c(P_k^* \setminus H_i) = c(P_k^1) + c(P_k^2) \leq 2c(P_k^1) \leq 2c(P_k^* \setminus H^*)$.

To conclude we prove that $\sum_{e \in H} c_e \leq 2 \sum_{e \in H^*} c_e$. Edges are only added to H during case 2 of the above procedure. Suppose this occurs with current component H_{i_1}, and with path P_k^* intersecting components $H_{i_1}, \ldots, H_{i_q}$ with $i_1 < \cdots < i_q$. By eligibility for case 2, the edges added to H at this point have cost at most $\sum_{s=2}^{q} \sum_{e \in H_{i_s}} c_e$. The key observations are these: only components with index larger than that of the current component appear in this expression ($i_s > i_1$ for $s > 1$); once a component appears in this expression, its edges are absorbed into the current component (which retains its index); edges of any such component lie in H^*; and the index of the current component can only increase. Because of these four facts, every edge of H^* participates in the expression $\sum_{s=2}^{q} \sum_{e \in H_{i_s}} c_e$ at most once. Summing over all additions of edges to H, we get $\sum_{e \in H} c_e = \sum_{e \in H^*} c_e + \sum_{e \in H \setminus H^*} c_e \leq 2 \sum_{e \in H^*} c_e$. ■

2.2 An LP formulation

We now give an integer programming formulation for MCFL. The decision variables are of the form x_{ij} (1 if demand j is assigned to facility i and 0 otherwise) and z_e (1 if e is selected as a connecting edge and 0 otherwise). The integer program is as follows:

$$\min \sum_{j \in \mathcal{D}} \sum_{i \in V} x_{ij} d(i,j) + M \sum_{e \in E} c_e z_e \quad \text{s.t.} \qquad \text{(IP)}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in \mathcal{D}$$

$$\sum_{e \in \delta(S)} z_e \geq \sum_{i \in S} x_{is_k} - \sum_{i \in S} x_{it_k} \qquad \forall S \subseteq V, s_k \in \mathcal{D}$$

$$\sum_{e \in \delta(S)} z_e \geq \sum_{i \in S} x_{it_k} - \sum_{i \in S} x_{is_k} \qquad \forall S \subseteq V, t_k \in \mathcal{D}$$

$$x_{ij}, z_e \in \{0,1\},$$

where $\delta(S)$ is the set of edges having precisely one endpoint in S. We replace the integrality constraint by $x_{ij}, z_e \geq 0$

for all e, i, j to obtain a linear program. The dual to this relaxation is

$$\max \quad \sum_{j \in \mathcal{D}} \alpha_j \quad \text{s.t.} \qquad \text{(DP)}$$

$$\alpha_{s_k} - \sum_{S:i \in S} y_{S,s_k} + \sum_{S:i \in S} y_{S,t_k} \le d(i, s_k) \qquad (1)$$

$$\alpha_{t_k} - \sum_{S:i \in S} y_{S,t_k} + \sum_{S:i \in S} y_{S,s_k} \le d(i, t_k) \qquad (2)$$

$$\sum_{j \in \mathcal{D}} \sum_{S:e \in \delta(S)} y_{S,j} \le M c_e \ \ \forall e \in E \qquad (3)$$

$$y_{S,s_k}, y_{S,t_k} \ge 0 \qquad (4)$$

where constraints (1) and (2) range over all $(s_k, t_k) \in \mathcal{D}$ and $i \in V$. By weak duality, any feasible solution to this dual LP is a lower bound on the cost of an optimal solution to the connected facility location problem.

The dual LP should be interpreted as follows. The value α_j is the amount that demand $j \in \mathcal{D}$ is "willing to pay" towards a solution. If demand j is assigned to facility i, a portion of α_j pays for the distance $d(i, j)$; the rest contributes to the connecting edges. At the highest level, the goal of our algorithm (and of any primal-dual algorithm) is to raise the dual variables α_j as much as possible ("generating revenue") while maintaining dual feasibility, thereby ensuring that $\sum_{j \in \mathcal{D}} \alpha_j$ is a valid lower bound on the optimum.

3 The Algorithm

3.1 Difficulties

Before presenting our algorithm, we try to indicate some of the main difficulties that arise in solving MCFL. We first propose a simple primal-dual algorithm for the problem. Call a demand j *tight with facility* i if the constraint (1) for j is satisfied with equality (with respect to the current dual solution), and edge e *tight* if the constraint (3) for e is satisfied with equality. Call a facility i *reachable from* j if there is a facility k with the following property: j is tight with k and there is a path of tight edges between k and i. The algorithm is as follows, and is similar to that of Jain and Vazirani [16] for classical facility location. We begin with all dual variables set to zero, and begin raising the α_j's at a uniform rate. We also raise the dual variable $y_{S_j,j}$ in conjunction with j, where S_j is set of facilities reachable from j. This procedure ensures dual feasibility with the constraints (1) and (2) replaced by

$$\alpha_{s_k} - \textstyle\sum_{S:i \in S} y_{S,s_k} \le d(i, s_k) \quad \forall\, s_k \in \mathcal{D}, i \in V \qquad (5)$$

$$\alpha_{t_k} - \textstyle\sum_{S:i \in S} y_{S,t_k} \le d(i, t_k) \quad \forall\, t_k \in \mathcal{D}, i \in V. \qquad (6)$$

We ignore further issues of dual feasibility for the moment (though our algorithm must handle this difficulty).

When M unassigned demands become tight with a common facility, we open the facility and call this group of demands a *cluster*. Assume we succeed in clustering all of the demands into groups of size M. Intuitively, these are groups large enough to justify building edges to connect the open facilities (since the cost of building an edge is M times the cost of assigning demands across an edge). These clusters induce an instance of the well-solved generalized Steiner problem [1, 10] with clusters as terminals and connectivity requirements induced in a natural way. This suggests running a primal-dual algorithm for the generalized Steiner problem (as in [1, 10]).[2] Unfortunately, a problem arises. The algorithms of [1, 10] build edges one-by-one, until all connectivity requirements are satisfied. When an edge is built, two components of edges merge into one; in our application, this may connect many of the demand pairs in the original connected facility location instance, dropping the number of unsatisfied demands in the new component to a nonzero number much less than M. We may thus encounter a partial solution that fails to satisfy all connectivity requirements and also fails to cluster unsatisfied demands into large enough groups to justify building further edges.

To handle this problem, we are forced to interleave clustering and building phases. This in turn causes several technical problems that must be dealt with. For example, in any given phase, the dual variables of previous phases will contribute to the constraints of type (3), thereby creating many tight edges and forcing the reachable sets S_j to grow large quickly. We deal with this problem in two ways.

First, we break our algorithm into stages, with the dual increase of each variable in one stage being a constant factor larger than the increase in the previous stage; this ensures that dual increases in one stage cannot affect future stages too much. Second, we introduce a method for bounding the proliferation of tight edges via a *distance-preserving property*. Roughly speaking, this property asserts that we can pay for "most" of the tight edges with the current dual solution, in the following sense: if T is the set of tight edges and $B \subseteq T$ are the edges that we can pay for with the current dual solution, then the distances between any pair of vertices in the graphs G_B and G_T obtained by contracting the edges of B and T, respectively, differ only by a small factor. We then show that all demand pairs with source and sink "not too far apart" can be assigned to facilities in the graph G_T with a cost that can be accounted for with our current dual solution; the smallest distance qualifying as "far apart" will increase exponentially with the number of stages. By the distance-preserving property, it follows that assignments in G_B of such demands can be (approximately) paid for. The cost of assignments in G_B approximately reflect the cost of assignments in G (since the contracted edges in G_B

[2]Indeed, this approach leads to a constant-factor approximation for the special case when all sinks are identical.

are the edges of B, which are already paid for and can therefore be used freely), so such demands can be assigned to open facilities without incurring too much cost. At the end of the algorithm, all demands are "not too far apart", and we obtain a feasible solution with small cost.

3.2 Some Preliminaries

Auxiliary Graphs. Our algorithm maintains two graphs, G' and G_B. Both of these graphs will change throughout the execution of our algorithm. Let $d_{G'}$ and d_{G_B} denote shortest-path distance in these two graphs. Let B denote the set of edges already built by our algorithm. As in the previous section, the graph G_B is obtained from G by contracting all edges in B. The distance $d_{G_B}(s_k, t_k)$ should be interpreted as the cost of assigning s_k and t_k to open facilities that are already connected to each other (this is not quite true, but motivates why G_B is a useful network to consider). Since the edges in B are in some sense "already paid for", G_B can be thought of as a "residual network". Also, each connected subgraph H of G_B corresponds to a connected subgraph of G in a natural way; we denote this subgraph by $G[H]$.

The graph G' intuitively corresponds to the graph G_T of the previous section, but has a more complicated definition. We will call G' the *auxiliary graph*. At every point in our algorithm, the graph G' is obtained from G_B by a sequence of the following two operations: (1) contract an edge of G_B; (2) decrease the length, $\ell(e)$, of an edge e from c_e to zero. (While setting an edge length to zero is intuitively the same as contracting it, it will be technically convenient to distinguish between these two operations.) The distance $d_{G'}$ in G' will be with respect to the length $\ell(e)$ of edges in G', which for each edge e will be either c_e or zero.

A vertex $v' \in G'$ corresponds to a connected subgraph in both G_B and in G—we denote these subgraphs $G_B[v']$ and $G[v']$, respectively (see Figure 1). We define $G_B[H']$ and $G[H']$ for a connected subgraph H' of G' in a similar manner. We intuitively think of G' as a "coarser version" of G_B, with each vertex v' in G' representing a small "region" in G_B. We associate with v' a vertex of $G_B[v']$ that we call a *core* (denoted $\mathrm{core}(v')$). As a vertex in G_B, $\mathrm{core}(v')$ is a connected component of built edges that we think of as being "nearby" all vertices of $G_B[v']$; thus if demands in v' need to be assigned to an open facility, $\mathrm{core}(v')$ represents some that are close by. Similarly, building edges between u' and v' in G' should translate in a distance-preserving manner to building edges between $\mathrm{core}(u)$ and $\mathrm{core}(v)$ in G_B.

Each vertex v of G or G_B is contained in some vertex of G'; we will denote this vertex by $v(G')$. When we speak of demand j in G_B (or G'), we mean the vertex of G_B (or G') that contains j.

Some Assumptions. We next make two easily-imposed assumptions about the problem input, which will simplify

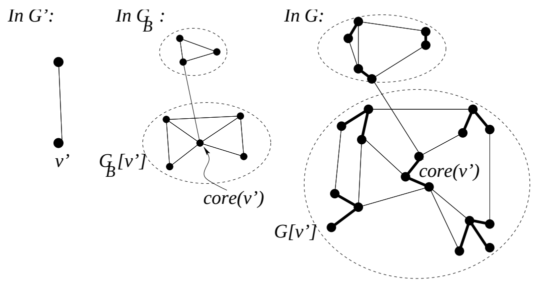

Figure 1. Mappings between vertices of various graphs. Bold edges have been built.

the description of our algorithm: (1) we assume that every edge with non-zero cost has cost precisely ϵ, where ϵ is a sufficiently small constant; (2) we assume that the distance between any two demands that are not co-located is at least $20C^2$, where C is a sufficiently large constant.

Assumption (2) can be enforced by scaling up all distances of the input graph. Assumption (1) is enforced by rounding edge costs to a multiple of ϵ and then subdividing edges until all edges have length precisely ϵ. This permits the dual increases in our algorithm to occur in discrete steps, and affects the approximation ratio by a negligible factor. These subdivisions allow for facilities to be located at these new subdividing points, but simple postprocessing relocates facilities at the original vertices at the expense of a small constant factor increase in the solution cost. There is also a concern that these subdivisions may result in a pseudopolynomial time algorithm, but in fact the algorithm we give below for subdivided networks is easily converted into a strongly polynomial time algorithm. Details are given in the full version.

Defining Tight. We now give our revised definition of what it means for a demand j to be *tight* with a facility i. The definition will be similar but not identical to the notion of "reachable" in Subsection 3.1, and will make use of the auxiliary graph G'.

Initially, a demand j is tight with all facilities that are co-located with it (including j itself). A demand j can become tight with additional facilities when its dual variable α_j is increased or when the auxiliary graph G' is modified. First, if α_j is raised by ϵ units (by assumption (1) above, all dual increases are of this form), then by definition j becomes tight with any facility i satisfying the following: there is a facility k of G and vertices i' and k' of G' containing i and k such that j is tight with k and $d_{G'}(k', i') = \epsilon$.

Finally, the facilities with which j is tight will satisfy the following closure property, by definition: If j is tight with a facility contained in vertex v' of G' and $d_{G'}(v', w') = 0$, then j is tight with all facilities of G contained in w'. This invariant implies that modifications of G' (contracting an edge or decreasing the length of an edge to zero) can implicitly increase the number of facilities with which a demand is tight. Note also that by taking $v' = w'$ in this invariant, it makes sense to say that a demand j is tight with a facility i' in G'; this simply means that j is tight with all facilities of G in $G[i']$.

3.3 Algorithm Description

Disclaimer: Our algorithm will not output a feasible solution to MCFL, but will instead output a partial solution that is easily transformed into a feasible solution with at most twice the cost. Our output consists of a set B of edges and for each demand pair (s_k, t_k), an s_k-t_k path P_k. Our algorithm will guarantee that $M \sum_{e \in B} c_e + \sum_{k=1}^{p} c(P_k \setminus B)$ is at most a constant factor larger than the cost of the optimal solution to the original connected facility location problem. We will say that edges in B are *bought* or *built* and that edges in $P_k \setminus B$ are *rented*. If we began with an instance of MROB, then it is straightforward to check that $P_1, \ldots, P_p$ can be reinterpreted as a solution for this instance with the same (or smaller) cost; it follows from the proof of Lemma 2.1 that this solution has cost within a constant factor of optimal for the rent-or-buy problem. If we desire a solution to the MCFL instance, then the reduction of Lemma 2.1 can be used to extract one from $P_1, \ldots, P_p, B$ that is within a constant factor of optimal.

First, we describe some initial conditions for our algorithm. We start with the empty primal solution (no edges built, no assignments made) and the all-zero dual solution. The auxiliary graph G' is initially G. Each vertex v' of G' will maintain a *budget* (intuitively, the amount of "revenue" it has raised to pay for building edges and making assignments); initially, all budgets are zero. Every demand will be in one of three states: (1) *alive and unfrozen*, which indicates an unassigned demand that is allowed to raise its dual variable; (2) *alive and frozen*, which indicates an unassigned demand that is not allowed to raise its dual variable; (3) *dead*, which indicates an assigned demand. Initially, every demand is alive and unfrozen; frozen demands may subsequently be unfrozen, but dead demands will never be resurrected. Every facility will be either *frozen* (if it participates in a cluster of M demands) or *unfrozen* (otherwise). Initially, every facility is unfrozen; frozen facilities may later become unfrozen.

Whenever we increase a dual variable α_j, we simultaneously increase the dual variable $y_{S_j,j}$, where S_j is the set of facilities with which j is tight (with the definition of tight given in the previous subsection); we will see that this ensures that the relaxed dual constraints (5) and (6) are always satisfied. We will henceforth only describe how to raise the α_j's, which we refer to as "the dual variables", with the understanding that the $y_{S_j,j}$'s are raised in this way.

We now describe stage r of our algorithm ($r \geq 0$). Let G^r be the auxiliary graph at the beginning of stage r; G' will always denote the auxiliary graph at the current point of time (so $G' = G^r$ at the beginning of stage r). The end of the previous stage defines a set F_r of frozen facilities of G^r (in stage 0, $F_0 = \emptyset$); all other facilities, as well as all alive demands, are unfrozen. The previous stage will ensure that facilities of F_r are far from each other in G^r, that each facility $i' \in F_r$ possesses a set $P(i')$ of M *primary demands* that are tight with i', and that no demand is a primary demand for two different frozen facilities. A stage of the algorithm consists of three phases.

Phase 1: The purpose of this phase is to form clusters. We implement this as follows. Dual variables corresponding to alive, unfrozen demands are raised uniformly, until one of the following events happen.

1. There is a frozen facility i' and an alive unfrozen demand j such that either $d_{G'}(i', j) \leq 8 \cdot C^{r+1}$ or j gets tight with i'. (Recall from the previous subsection that C is a sufficiently large constant.) Freeze the demand j and stop raising α_j.

2. There are M alive demand nodes tight with a facility i', and not all of them are frozen. Freeze the facility i', add it to F_r, and set the M demand nodes to be $P(i')$, the primary demands of i'.

3. There is a demand j such that α_j is at least C^{r+1}. Freeze the demand j and stop raising α_j.

Note that freezing of demands or facilities may occur before any dual variables have been raised. Phase 1 terminates when all alive demands are frozen. If more than one of these three events happen simultaneously, we give precedence to event (1).

Phase 2: This phase increases dual variables further (while still ignoring the issue of dual feasibility) to pay for assigning demands and building edges later in this stage. Precisely, we increase α_j of every demand j in $\cup_{i' \in F_r} P(i')$ by C^{r+1}. The budget of each node in F_r is updated to be C^{r+1}.

Phase 3: The final phase of our algorithm is the most complicated and breaks down into several procedures. We maintain a set of nodes X which is initially set to F_r. Let Z_r be the set of edges in G^r ($= G'$) such that the constraint (3) corresponding to these edges in the dual LP is violated, and set $\ell(e)$ to 0 for all these edges. (Of course, $\ell(e)$ may already be 0 in G^r for some edges). Let G' be this new auxiliary graph. Note that G^r and G' have the same set of vertices, but the distance functions in the two graphs are different.

Setting lengths of some edges to be 0 can contract distances in G' by a lot, compared to distances in G_B. In the procedure **CreateNodes**, we identify places where distances have contracted substantially. Since our aim is to maintain the fact that two points are nearly at the same distance in G_B as in G', we build edges in G_B at some of these places so that the corresponding distances go down in G_B as well. To this end, we add more vertices to the set X. To each vertex $v' \in X$, we associate a subgraph $\mathbf{B}(v')$ of G', which is the set of all nodes within distance at most $11 \cdot C^{r+1}$ of v' in G^r. Note the subtlety here that the distance is measured in G^r and not in the current auxiliary graph G'.

Procedure CreateNodes: Suppose there are two vertices $u', v' \in G'$ and P' is a shortest path between them in G' (according to ℓ, the length function on G'). Let $\gamma \ll C$ be a sufficiently large constant. Further suppose that u', v', P' satisfy the following properties: (1) none of the points in P' belong to any of the balls $\mathbf{B}(w')$ for any $w' \in X$; and (2) $d_{G_B}(\text{core}(u'), \text{core}(v'))$ is between γC^{r+1} and $2\gamma C^{r+1}$, whereas $d_{G'}(u', v') \leq \gamma C^{r+1}/4$.

We choose a set of γ points $u'_0, u'_1, \ldots, u'_\gamma$ from the path P' as follows: $u'_0 = u'$, u'_1 is the right-most point on P' such that $(1 - 1/4) \cdot C^{r+1} \leq d_{G_B}(\text{core}(u'), \text{core}(u'_1)) \leq C^{r+1}$, u'_2 is the right-most point in P' such that $d_{G_B}(\text{core}(u'), \text{core}(u'_2))$ is between $(2 - 1/4) \cdot C^{r+1}$ and $2 \cdot C^{r+1}$ and so on. We stop when we find γ such points. Existence of these points and the fact that they lie on P' in this order are proved in the full version. Let $\mathcal{D}(u', v')$ denote the set of these points, and add these $|\mathcal{D}(u', v')| = \gamma$ points to X. We shall say that this procedure creates the pair (u', v').

As before, we also construct the balls $\mathbf{B}(u'_l)$ around all $u'_l \in \mathcal{D}(u', v')$. Note that the union of these balls may not cover all of P'. Indeed, since shortest paths in G' do not map to shortest paths in G_B, there may be a point between u' and u'_1 whose distance from $\text{core}(u')$ in G_B is much more than C^{r+1}.

We keep doing this operation above as long as it is possible. At the end, for each $v' \in X$, we want to contract the sets $\mathbf{B}(v')$ into single nodes. The first problem with this is more of a technical issue. For $w' \in \mathbf{B}(v')$, look at the shortest path in G_B joining $\text{core}(w')$ and $\text{core}(v')$; all the edges in this path may not lie in the set $\mathbf{B}(v')$. To handle this, we *complete* the set $\mathbf{B}(v')$ to $\mathbf{B}'(v')$ thus: initially $\mathbf{B}'(v')$ contains just $\mathbf{B}(v')$. Now if there is a vertex $w' \in \mathbf{B}(v')$ such that the shortest path between $\text{core}(v')$ and $\text{core}(w')$ in G_B uses a vertex x, where $x(G')$ is not in $\mathbf{B}(v')$, then we add $x(G')$ to $\mathbf{B}'(v')$.

We now want to contract $\mathbf{B}'(v')$ into a single node. Another problem presents itself: If $\mathbf{B}'(v')$ and $\mathbf{B}'(u')$ for $u' \neq v' \in X$ share some vertices, then both sets will get contracted to the same node. To decide what the core of this new node will be, we run the following procedure :

Procedure ContractTree(X): Let us construct a graph G_X on the vertex set X thus: $u', v' \in X$ are joined by an edge if $\mathbf{B}'(u') \cap \mathbf{B}'(v') \neq \emptyset$. Now let T_X be a spanning forest in G_X; i.e., T_X restricted to any connected component of G_X is a spanning tree. For each edge $e = (u', v') \in T_X$, let $w' \in \mathbf{B}'(u') \cap \mathbf{B}'(v')$. Find the shortest path between $\text{core}(u')$ and $\text{core}(w')$ in G_B that lies entirely inside $G_B[\mathbf{B}'(u')]$. Similarly, find a path from $\text{core}(w')$ to $\text{core}(v')$ in G_B. Build edges on these paths (hence adding these edges to the set B as well, and contracting all these edges in G_B).

Contract all the vertices in $\mathbf{B}'(v')$ to a single node for each $v' \in X$ in the auxiliary graph. If x' is such a node, then x' may have been obtained by contraction of several of the sets $\mathbf{B}'(v'_1), \ldots, \mathbf{B}'(v'_s)$, where $v'_1, \ldots, v'_s$ form a connected component of G_X in the procedure above. However, note that these contractions are accompanied by the building of edges, and hence $\text{core}(v'_1), \ldots, \text{core}(v'_s)$ contract to a single node in G_B as well. This is defined as $\text{core}(x')$ in G_B, and we set the budget of x' to be C^{r+1}.

Procedure Contract: As the last round of building edges in a stage, we perform the following operation as long as possible. Let u', v' be two nodes with budget C^{r+1} such that the shortest path P' between them in G' has no internal vertex of budget C^{r+1}. Furthermore, suppose that these nodes are "somewhat close"; i.e., $d_{G'}(u', v') \leq 9 \cdot C^{r+2}$. Let $u = \text{core}(u'), v = \text{core}(v')$, find a shortest path P in G_B between u and v, and build edges on this path. Furthermore, if P contains a vertex w such that $w(G')$ is a vertex of budget C^{r+1} in G', then find a shortest path in $G_B[w']$ between w and $\text{core}(w')$, and build edges on this path as well. Contract the edges we just built in G_B. Note that $\text{core}(u')$ and $\text{core}(v')$ will contract to a single vertex in G_B, call this x. P corresponds to a path P'' joining u' and v' in G', and contracting all the edges in P'' creates the new vertex x' in G'. We define $\text{core}(x') = x$, and allot x' a budget of C^{r+1}.

Procedure Prune Demands: Our next step in this phase is to satisfy demands that are sufficiently close to each other in G'. Formally, let s_k, t_k be a pair of alive demands with $d_{G'}(s_k, t_k) \leq 5 \cdot C^{r+2}$; define path P_k connecting them to be the shortest s_k-t_k path in G_B, lifted to an s_k-t_k path of G in the obvious way. Edges of $P_k \setminus B$ are rented, and demands s_k and t_k are marked dead (we will never consider them again in the algorithm).

Procedure Regrow: The final procedure of Phase 3 raises the duals α_j of some demand nodes. For a vertex v' with a budget of C^{r+1}, define $D(v')$ to be the set of those demands j that are only tight with $G[v']$. In particular, such a demand j must lie in $G[v']$, because every demand is tight with itself.

If there is a vertex v' of budget C^{r+1} such that $|D(v')| < M$, then we start raising the α_j value of all demands in $D(v')$ simultaneously, stopping an α_j from rising further if it reaches C^{r+1}. Define F_{r+1} to be the set of all nodes v' that have a budget of C^{r+1}, and that also satisfy $|D(v')| \geq M$. This is the set of frozen facilities for the next stage. Furthermore, for $v' \in F_{r+1}$, define the primary demands $P(v')$ of v' to be any M of the demands in $D(v')$.

4 Overview of Analysis

We now give a high-level overview of the analysis; the precise arguments are given in the full version.

Approximate Dual Feasibility. Unlike a traditional primal-dual algorithm, our algorithm does not explicitly maintain feasibility of the dual solution. We explicitly obey the relaxed constraints (5) and (6) and do not obey the dual constraint (3) for edges at all. On the other hand, we prove that our algorithm always maintains a dual solution that is *approximately* optimal.

Theorem 4.1 *If (α, y) is the dual solution produced by the algorithm, then $(\frac{1}{5}\alpha, \frac{1}{5}y)$ is feasible for the LP (DP).*

Theorem 4.1 is proved in two steps. First, we use the fact that α_{s_k} and α_{t_k} are only raised when s_k and t_k are far apart (because of the **Prune Demands** procedure) in conjunction with feasibility for constraints (5) and (6) to show that constraints (1) and (2) are satisfied. Second, we show that no dual constraint of the form (3) is violated by more than a factor 5. Since we give edges with violated dual constraint length 0 at the beginning of Phase 3 (after which the left-hand side of the constraint will never increase again, by definition of tight), it suffices to prove that the contribution to the left-hand side of the dual constraint corresponding to edge e in a single stage is at most $4Mc_e$. It is easy to show that a single demand can contribute only c_e to the left-hand side, so the problem reduces to showing that only $O(M)$ demands contribute to the left-hand side. Our algorithm ensures this property by only allowing M different demands to become tight with a vertex (this limits the contribution in Phase 1), and by forcing any two frozen facilities to be far apart; primary demands are relatively close to their frozen facilities, and therefore primary demands belonging to different facilities cannot contribute to a single dual constraint (this limits the contribution in Phase 2).

The Distance-Preserving Property. We next argue that distances in G_B and G' are close to each other during the entire run of the algorithm. Let β and λ be constants such that $\beta \ll \gamma \ll \lambda \ll C$.

Theorem 4.2 *Let v' be a node with budget C^s, and let v be its core in G_B. Then $d_{G_B}(u, v) \leq \beta C^s$ for any $u \in G_B[v']$, with such a path lying inside the subgraph $G_B[v']$.*

Furthermore, let $u', v' \in G^s$, with cores $u, v \in G_B$ respectively. Then $d_{G_B}(u, v) \leq 5 \cdot d_{G^s}(u', v') + \lambda C^s$.

We prove Theorem 4.2 by induction on s. Assume the theorem holds for all stages before r. In stage r, we construct the balls $\mathbf{B}(v')$ for each $v' \in X$ of radius about C^{r+1} in G^r, which by induction correspond to radius $O(C^{r+1})$ balls in G_B, as well. Since we construct nodes of budget C^{r+1} by collapsing such balls, the first part of the inductive step essentially follows from the fact that these balls have radii $O(C^{r+1})$ in G_B.

Now we show how to prove the second part of the theorem above. Suppose we have finished the procedure **ContractTree** in Stage r. Let $u', v' \in G'$ have a shortest path P' between them in G' that does not contain any vertex of weight C^{r+1}, and $d_{G_B}(\mathrm{core}(u'), \mathrm{core}(v'))$ lies between γC^{r+1} and $2\gamma C^{r+1}$. We claim that $d_{G'}(u', v') \geq \gamma C^{r+1}/4$; indeed, otherwise we would have considered this path in the procedure **CreateNode** and collapsed it. If u', v', P' satisfy the above properties but $d_{G_B}(\mathrm{core}(u'), \mathrm{core}(v'))$ is much bigger than γC^{r+1}, then we can break the path P' into smaller segments of length about γC^{r+1} and argue independently on each segment. Thus the distance between u' and v' is nearly the same in G_B and G'; though this is only up to an additive factor of about γC^{r+1}.

But what if P' contains nodes of budget C^{r+1}? These nodes can be a problem: since they correspond to subgraphs of radii about C^{r+1} in G_B, we may be contracting distances substantially in G' by collapsing such subgraphs. Consider two consecutive nodes of budget C^{r+1} in P' — the procedure **Contract** ensures that the distance between them is so large that the contraction of these nodes will not have much effect on the distance between u' and v'. This allows us to prove the theorem.

The Performance Guarantee. The cost of the primal solution is the sum of the cost of renting some edges and the cost of building others. Let us first account for the rental costs: if we rent edges between s_k and t_k in the **Prune Demands** procedure of stage r, the distance between s_k and t_k is about C^{r+1} in G', and about the same in G_B as well (using the distance preserving property). So if α_{s_k} or α_{t_k} is at least C^{r+1} then these demands can pay for the cost of renting by their dual variables. If both α_{s_k} and α_{t_k} are small, we can show that both these demands are close enough to some frozen facility such that we can pay for renting of edges to this facility.

Accounting for the edges we build is more involved. Here we use the budgets of nodes, which is roughly the amount it can pay for building edges (scaled down by M). This explains why we assign a budget of C^{r+1} in Phase 2, since we can account for this by the raising of M of the α_j values. In **Contract**, since we build edges between two nodes of budget C^{r+1}, one of these high-budget nodes can pay for cost of building, which is $O(MC^{r+1})$. In procedure **ContractTree** also, each edge of the tree T_X basically corresponds to building edges on $O(C^{r+1})$ length paths, and so each node of T_X has to account for about C^{r+1} length edges. If a node of T_X comes from Phase 2, we know it has a budget of C^{r+1}, and so it can pay for building the edges. If not, then this node in T_X comes from the procedure **CreateNode**, and exists because of a pair (u', v') created by this procedure. But then the distance between u' and v' in G' was much less than that in G_B, and so many of the duals must have been raised for this shrinking of distances. We then show how to borrow C^{r+1} units of budget from these dual variables, which completes the proof of the following theorem.

Theorem 4.3 *The cost of the primal solution constructed by our algorithm is within a constant of $\sum_j \alpha_j$.*

Acknowledgments. We thank Jon Kleinberg and Éva Tardos for helpful discussions.

References

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

[2] M. Andrews and L. Zhang. Approximation algorithms for access network design. *Algorithmica*, 34(2):197–215, 2002.

[3] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proceedings of 38th FOCS*, pages 542–547, 1997.

[4] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized Steiner problem. In *Proceedings of 7th SODA*, pages 68–74, 1996.

[5] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of 30th STOC*, pages 161–168, 1998.

[6] Y. Bartal, M. Charikar, and P. Indyk. On page migration and other relaxed task systems. In *Proceedings of 8th SODA*, pages 43–52, 1997.

[7] N. Garg, R. Khandekar, G. Konjevod, R. Ravi, F. S. Salman, and A. Sinha. On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design formulation. In *Proceedings of 8th IPCO*, pages 170–184, 2001.

[8] M. X. Goemans and D. Bertsimas. Survivable networks, linear programming relaxations, and the parsimonious property. *Mathematical Programming*, 60:145–166, 1993.

[9] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of 5th SODA*, pages 223–232, 1994.

[10] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[11] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 4, pages 144–191. PWS Publishing Company, 1997.

[12] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proceedings of 41st FOCS*, pages 603–612, 2000.

[13] S. Guha, A. Meyerson, and K. Munagala. A constant factor approximation for the single sink edge installation problems. In *Proceedings of 33rd STOC*, pages 383–388, 2001.

[14] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *Proceedings of 33rd STOC*, pages 389–398, 2001.

[15] K. Jain, M. Madian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of 34th STOC*, pages 731–740, 2002.

[16] K. Jain and V. V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *Journal of the ACM*, 48:274–296, 2001.

[17] D. R. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of 41st FOCS*, pages 613–623, 2000.

[18] S. Khuller and A. Zhu. The general Steiner tree-star problem. *Information Processing Letters*, 2002. To appear.

[19] T. U. Kim, T. J. Lowe, A. Tamir, and J. E. Ward. On the location of a tree-shaped facility. *Networks*, 28(3):167–175, 1996.

[20] M. Labbé, G. Laporte, I. Rodrígues Martin, and J. J. Salazar González. The median cycle problem. Technical Report 2001/12, Department of Operations Research and Multicriteria Decision Aid at Université Libre de Bruxelles, 2001.

[21] Y. Lee, S. Y. Chiu, and J. Ryan. A branch and cut algorithm for a Steiner tree-star problem. *INFORMS Journal on Computing*, 8(3):194–201, 1996.

[22] M. Madian, E. Markakis, A. Saberi, and V. V. Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Proceedings of 4th APPROX*, pages 127–137, 2001.

[23] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM Journal on Optimization*, 11(3):595–610, 2000.

[24] C. Swamy and A. Kumar. Primal-dual algorithms for the connected facility location problem. To appear in *APPROX 2002*.

[25] K. Talwar. Single-sink buy-at-bulk LP has constant integrality gap. In *Proceedings of 9th IPCO*, pages 475–486, 2002.

[26] D. P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming, Series B*, 91(3):447–478, 2002.

[27] D. P. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15:435–454, 1995.

A Proofs

A.1 Feasibility

In this section, we argue that the dual variables $\alpha_j, y_{S,j}$ constructed by the algorithm of Subsection 3.3 are nearly feasible. We will first show that the constraints (5) and (6) are always satisfied. For this, we need some simple lemmas about the dual variables.

Lemma A.1 *At the beginning of stage r, $\alpha_j \le 2C^r$ for all demand nodes j.*

Proof: By induction of the number of stages. In Phase 1, no α_j is raised above C^{r+1}. In Phase 2, α_j is increased by at most an additive C^{r+1} factor. Finally, in **Regrow**, we never raise α_j above C^{r+1}. ■

Lemma A.2 *If j is tight with i' in the auxiliary graph G', then $d_{G'}(j, i') \le \alpha_j$ at that point in time.*

Proof: The lemma clearly holds at the beginning of the algorithm, so suppose the lemma is true at time t. Since distances in G' can only get smaller, we need only be concerned with the first moment in time at which a demand j becomes tight with some facility i' of G'.

We now look at the operations that algorithm performs. If it contracts a set of nodes S to a single node s, j will be tight with s only if j was tight with a node i in S, and hence by the induction hypothesis, $d_{G'}(j, s)$ will be bounded above by α_j. A similar argument applies when we set an edge length to 0.

Consider the point when time becomes $t+\epsilon$, and we raise α_j by ϵ. Suppose j becomes tight with i' at time $t+\epsilon$, but it was not tight with it at time t. By the definition of tight and the fact that all edges have length ϵ, this can only happen if j was tight with some node i'' at time t and $d_{G'}(i', i'') = \epsilon$. By induction, $\alpha_j \ge d_{G'}(i'', j)$ at time t, and thus at time $t+\epsilon$, $\alpha_j \ge d_{G'}(i', j)$, completing the proof. ■

Lemma A.3 *If j is not tight with facility i in G, then $\alpha_j < d(i, j)$.*

Proof: This is clearly true at the beginning of the algorithm. If the algorithm contracts some vertices into a single node or sets the length of an edge to 0, node j can only become tight with even more facilities. So suppose the algorithm raises the value of α_j by ϵ. Let $i' = i(G')$, and let $N(i)$ be the set of nodes l in G such that $d_{G'}(i', l(G')) \le \epsilon$. We consider two cases. First suppose that before α_j is raised, j is tight with a node in $N(i)$. In this case, after α_j is raised, it will become tight with i, maintaining the invariant. In the other case, α_j is not tight with any node in $N(i)$ before it is raised. In this case, the invariant implies that $\alpha_j < d(i'', j)$ for all $i'' \in N(i)$. Now there must be at least one $i'' \in N(i)$ such that $d(i'', j) = d(i, j) - \epsilon$, and so $\alpha_j < d(i, j) - \epsilon$. Since α_j increases by ϵ, it follows that its new value is less than $d(i, j)$, maintaining the invariant. ■

This proves that at the instant j becomes tight with a facility i, $\alpha_j \le d(i, j)$. From this point on, the left hand side of (5) for i and j will remain unchanged, and hence satisfied. (A symmetric argument holds for (6).) We next show that the original constraints (1), (2) are also satisfied.

For the next lemma, by the *active phase* of the algorithm we mean the operations that modify the dual variables (Phases 1 and 2 and procedure **Regrow**).

Lemma A.4 *Consider a point of time in the active phase. If G' is the current auxiliary graph, and s_j, t_j are a pair of alive demands, then $d_{G'}(s_j, t_j) \ge \alpha_{s_j} + \alpha_{t_j}$.*

Proof: The inequality is easily seen to hold in Phases 1 and 2 of stage 0, since we assume that $d_G(s_j, t_j) \ge 20C^2$ for all j. So fix a stage $r \ge 1$ and suppose s_j, t_j survived the **Prune Demand** procedure of stage $(r-1)$. It follows that $d_{G'}(s_j, t_j) > 5 \cdot C^{r+1}$ at this moment in time. Lemma A.1 implies that throughout stages $r-1$ and r, $\alpha_{s_j} + \alpha_{t_j} \le 4C^{r+1}$. Since distances in G' are not modified during the **Regrow** procedure of state $r-1$ and Phases 1 and 2 of stage r, the desired inequality holds throughout. ■

Theorem A.5 *The constraints (1) and (2) are always satisfied.*

Proof: We will provide the argument for constraints (1), as the argument for constraints (2) is symmetric. For contradiction, let t be the first time at which constraint (1) is not satisfied for demand s_j and facility i. Since the algorithm enforces the corresponding constraint (5), we can conclude that $y_{S,t_j} > 0$ for some S containing i. This, in turn, implies that t_j is tight with i. Look at the first time when t_j becomes tight with i. Lemma A.2 implies that $\alpha_{t_j} \ge d_{G'}(t_j, i(G'))$ where G' is the auxiliary graph at this point of time. Since the distances in the auxiliary graph can only get smaller and from this point onwards, and α_{t_j} and $\sum_{S:i\in S} y_{S,t_j}$ will be raised at the same rate, it follows that at time t,

$$\alpha_{t_j} - \sum_{S:i\in S} y_{S,t_j} + \sum_{S:i\in S} y_{S,s_j} \ge d_{G''}(i, t_j)$$

where G'' is the auxiliary graph at time t. We already know that

$$\alpha_{s_j} - \sum_{S:i\in S} y_{S,s_j} + \sum_{S:i\in S} y_{S,t_j} > d(i, s_j) \ge d_{G''}(i, s_j),$$

since (1) is violated. But adding the two inequalities above contradicts Lemma A.4. ■

Before we prove that (3) is approximately satisfied, let us prove a supporting lemma.

Lemma A.6 *Suppose i'_1, i'_2 are two frozen facilities in Phase 2 of stage r. Let $j_1 \in P(i'_1)$ and $j_2 \in P(i'_2)$. Then, $d_{G^r}(j_1, j_2) \geq 5C^{r+1}$.*

Proof: Lemma A.2 implies that $d_{G^r}(j_1, i'_1) \leq \alpha_{j_1}$ and $d_{G^r}(j_2, i'_2) \leq \alpha_{j_2}$, where $\alpha_{j_1}, \alpha_{j_2}$ are the dual variables at the end of Phase 2. Suppose i'_1 and i'_2 were frozen at the end of Phase 3 of stage $r-1$ and they survived procedure **Contract** in stage $r-1$, we know that $d_{G^r}(i'_1, i'_2) \geq 9C^{r+1}$. Since $\alpha_{j_1}, \alpha_{j_2} \leq 2C^{r+1}$, we get that $d_{G^r}(j_1, j_2) \geq 5C^{r+1}$.

Now consider the case that at least one of i'_1 and i'_2 were not frozen at the beginning of stage r. In Phase 1, several events may happen at the same time. But even then we can talk about an event occurring *before* another event simply because we shall deal with these events in some order. Let E_1 be the event when i'_1 freezes and E_2 be the event when i'_2 freezes. Suppose E_1 occurs before E_2. By definition of freezing of a facility, at least one demand $j_2 \in P(i'_2)$ was not frozen when E_2 occurs. It must be the case that $d_{G^r}(j'_2, i'_1) \geq 8C^{r+1}$. Otherwise, E_1 has already occurred, i.e., i'_1 is frozen and so we should have frozen j'_2 (note that this gets preference over E_2 – this is how we defined the ordering of events in Phase 1). This is a contradiction.

Hence, $d_{G^r}(j'_2, i'_1) \geq 8C^{r+1}$. Since any $\alpha_j \leq C^{r+1}$ in Phase 1, it must be the case that $d_{G^r}(i'_1, i'_2) \geq 7C^{r+1}$. Finally, since $d_{G^r}(j_1, i'_1), d_{G^r}(j_2, i'_2) \leq C^{r+1}$, this implies that $d_{G^r}(j_1, j_2) \geq 5C^{r+1}$. ■

Corollary A.7 *Suppose i'_1, i'_2 are two frozen facilities in Phase 2. Then $P(i'_1)$ and $P(i'_2)$ are disjoint.*

Theorem A.8 *At any instant in time, $\sum_{j \in \mathcal{D}} \sum_{S: e \in \delta(S)} y_{S,j} \leq 5Mc_e$.*

Proof: Fix an edge e and consider the above constraint (3) for this edge. Since we never raise $y_{S,j}$ where $e \in \delta(S)$ when $\ell(e) = 0$ in the auxiliary graph, we need to estimate the contribution to the left hand side till $\ell(e)$ is set to 0.

Note that it is enough to show that for any stage r, the contribution to the left hand side from Phase 3 of stage $r-1$ and Phases 1 and 2 of stage r is at most $4Mc_e$. Indeed, if $\ell(e) \neq 0$ at the end of Phase 2 of stage $r-1$, the left hand side was $< Mc_e$, and the increases in the following three stages can only bring the total to $5Mc_e$, proving the theorem.

First we show that the contribution to the left hand side in Phase 1 is at most $2Mc_e$. Let $e = (u', v')$ be an edge in the auxiliary graph $G' = G^r$. We increase $y_{S,j}$ occurring on the left hand side only if j is tight with exactly one of u' and v', say u'. Consider the time when j becomes tight with u'. If α_j is raised by c_e more units, j also becomes tight with v'; hence any demand contributes to the left hand side by at most c_e units.

We claim that there can be at most M demands which can contribute to the left hand side of e by being tight with u' but not with v'. Suppose there are at least M demands which contributed in this manner. Consider the demand j which contributed last (if there are several such demands, pick one arbitrarily). Suppose it began raising its α_j at time t. At time t, M demands are tight with u'. Further, j is not frozen (otherwise we will not increase its dual variable). But then u' should be frozen, and so we should not have raised α_j — a contradiction.

Similarly, at most M such contributions can be due to being tight with v' but not u', bringing the total to at most $2Mc_e$.

Note another simple fact proved by very similar arguments as above: for any demand j and edge e, $\sum_{S: e \in \delta(S)} y_{S,j} \leq c_e$. We can now show that the total contribution to an edge in Phase 2 is at most Mc_e. Let j_1, j_2 be as in Lemma A.6, and let j_1 be tight with i''_1, and j_2 with i''_2 in Phase 2. Since $d_{G^r}(i''_1, j_1), d_{G^r}(i''_2, j_2) \leq 2C^{r+1}$, Lemma A.6 implies that $d_{G^r}(i''_1, i''_2) \geq C^{r+1}$. But since $c_e \leq \epsilon$ for any edge e, this means that j_1 and j_2 cannot both contribute to the left hand side of constraint (3) for e in this phase. Hence, the left hand side of the constraint for e can only get contributions from $P(i')$ for some fixed frozen i', which is at most Mc_e.

Finally, the argument for Phase 3 of stage $r-1$ is exactly the same as for Phase 2 of stage r, giving a contribution of Mc_e, bringing the grand total to $4Mc_e$, and hence proving the theorem. ■

A.2 Distance Preserving Property

We shall prove the following facts by induction on s:

- Let v' be a node with budget C^s, and let $v \in G_B[v']$ be the core of v' in G_B. If $u \in G_B[v']$ be any other vertex in $G_B[v']$, then $d_{G_B}(u, v) \leq \beta C^s$. Furthermore, a path of length at most βC^s can be found inside the subgraph $G_B[v']$.

- Let u', v' be two nodes in G^s, and let $u, v \in G_B$ with $u = \text{core}(u'), v = \text{core}(v')$. Then $d_{G_B}(u, v) \leq 5 \cdot d_{G^s}(u', v') + \lambda C^s$.

Here β and λ are large enough constants, satisfying $\beta \ll \gamma \ll \lambda \ll C$. Assuming the invariants hold for all $s \leq r$, we will show that they hold for $s = r+1$. Since nodes with budget C^{r+1} are created only in stage r, we need to look at the algorithm in stage r.

Theorem A.9 *Let v' be a node with budget C^{r+1} at the end of stage r, and let w be a node in $G_B[v']$. Then $d_{G_B}(w, \text{core}(v')) \leq \beta C^{r+1}$.*

Proof: It suffices to show that if $u' \in X$ in Phase 3 (before the procedure **TreeGrow**) and $v \in G_B[\mathbf{B}'(u')]$, then $d_{G_B}(\mathrm{core}(u'), v) \leq \beta C^{r+1}$. Indeed, suppose this is true. In procedure **ContractTree**, consider the case that x' is formed by contracting $\mathbf{B}'(u'_1), \ldots, \mathbf{B}'(u'_s)$. Then $\mathrm{core}(x')$ contains $\mathrm{core}(u'_1), \ldots, \mathrm{core}(u'_s)$, and $G_B[x'] = G_B[\mathbf{B}'(u'_1)] \cup \cdots \cup G_B[\mathbf{B}'(u'_s)]$. If we prove the result for elements in X, it will hold at the end of procedure **ContractTree** as well.

Now, suppose the theorem holds before we apply the procedure **Contract** on vertices u' and v' in G' to find a path P' between u' and v' and contract it to a node x'. Let $w \in G_B[x']$. It must be the case that $w' = w(G')$ is in P'. If budget of w' is C^{r+1}, then $\mathrm{core}(x)$ also contains $\mathrm{core}(w')$, and the theorem holds. If budget of w' is less than C^{r+1}, then some vertex $w_1 \in G_B[w']$ must be in the path P constructed to join $\mathrm{core}(u')$ and $\mathrm{core}(v')$. Since P is in $\mathrm{core}(x')$, $d_{G_B}(w, \mathrm{core}(x')) \leq d_{G_B}(w, w_1) \leq d_{G_B}(w, \mathrm{core}(w')) + d_{G_B}(w_1, \mathrm{core}(w')) \leq 2\beta C^r \leq \beta C^{r+1}$.

Hence it is enough to show the theorem only for sets $\mathbf{B}'(u')$. First, let us show this only for the set $\mathbf{B}(u')$ constructed for a node $u' \in X$, and we show it for the rest of the vertices in $\mathbf{B}'(u') - \mathbf{B}(u')$ later. Recall that if $w' \in \mathbf{B}(u')$, then $d_{G^r}(u', w') \leq 11 \cdot C^{r+1}$. Since w' is a node in G^r, it has a budget of at most C^r in G^r. Hence for $x \in G_B[w']$, the induction hypothesis implies that:

$$d_{G_B}(\mathrm{core}(u'), x) \leq d_{G_B}(\mathrm{core}(u'), \mathrm{core}(w')) + d_{G_B}(\mathrm{core}(w'), x) \leq 5 \cdot 11 \cdot C^{r+1} + \lambda C^r + \beta C^r \leq \beta/2 C^{r+1}. \quad (7)$$

Now we consider the other case, i.e., when $x \in G_B[\mathbf{B}'(u')] - G_B[\mathbf{B}(u')]$. It must be the case that $x' = x(G') \in \mathbf{B}'(u') - \mathbf{B}(u')$. So x' was added to $\mathbf{B}'(u')$ because there is a node $y \in G_B[\mathbf{B}(u')]$ and a node $z \in G_B[x']$ such that $d_{G_B}(\mathrm{core}(u'), z) \leq d_{G_B}(\mathrm{core}(u'), y) \leq \beta/2C^{r+1}$. So $d_{G_B}(\mathrm{core}(u'), x) \leq \beta/2C^{r+1} + d_{G_B}(x, z) \leq \beta/2C^{r+1} + 2\beta C^r \leq \beta C^{r+1}$. This proves the theorem, and the first part of the induction hypothesis. ∎

We now go on to the second part of the induction. Let G' be an auxiliary graph during the run of Procedure **Contract** in Phase 3. Note that G' may change during this procedure, so let us fix any such G'.

Before we begin the proof, let us give a mapping of paths between vertices in G' to paths in G_B that connect their cores. Formally, given vertices $u', v' \in G'$ and a path P' connecting them, $\mathcal{P}(u', v', P')$ specifies a path P in G_B which joins $\mathrm{core}(u)$ and $\mathrm{core}(v)$. This path P contains all the edges in P'; furthermore, if $e' = (u'_1, v'_1) \in P'$ corresponds to an edge $e = (u_1, v_1)$ in G_B, then P also contains edges that join u_1 to $\mathrm{core}(u'_1)$, and v_1 to $\mathrm{core}(v'_1)$ by shortest paths in $G_B[u'_1]$ and $G_B[v'_1]$ respectively.

Lemma A.10 *Let G' be the auxiliary graph at some time during the execution of procedure* **Contract**. *Let u', v' be nodes in G' with $u = \mathrm{core}(u')$ and $v = \mathrm{core}(v')$. If there is a shortest path P' between u' and v' in G' with no internal node of P' having a budget of C^{r+1}, then $d_{G_B}(u, v) \leq 4d_{G'}(u', v') + \lambda/2C^{r+1}$.*

Proof: Suppose no (internal or external) node of P' has budget C^{r+1}. Let $P = \mathcal{P}(u', v', P')$; note that P need not be a shortest path between u and v in G_B. Given vertices x, y on P, let P_{xy} be the segment of P between them.

Let $u'_1 = u', u_1 = u$. Starting from the node u, let w_2 be the first node on P such that $d_{G_B}(u_1, w_2) \geq 3/2\gamma C^{r+1}$. Since all edges are of length ϵ, $d_{G_B}(u_1, w_2)$ will be nearly $3/2\gamma C^{r+1}$. This node w_2 will be part of some node $u'_2 \in G'$, i.e., $w_2 \in G_B[u'_2]$; let $u_2 \in G_B$ be the core of u'_2. By the construction of $\mathcal{P}(u', v', P')$, u_2 lies on P as well. Furthermore, the distance from w_2 to u_2 in G_B will be at most $\beta C^r \leq C^{r+1}/2$ by Theorem A.9, and since $\gamma \geq 1$, $d_{G_B}(u_1, u_2)$ is between γC^{r+1} and $2\gamma C^{r+1}$. Continuing this way, we can find nodes $u'_1 = u', u'_2, \ldots, u'_k = v'$ on P' (with $u_j = \mathrm{core}(u'_j)$ for all j) such that $d_{G_B}(u_i, u_{i+1})$ lies between γC^{r+1} and $2\gamma C^{r+1}$ for all i, except possibly the last segment (which may violate the lower bound, but still satisfies $d_{G_B}(u_{k-1}, u_k) \leq 2\gamma C^{r+1}$).

We now claim that if $s < k - 1$, then $d_G(u_s, u_{s+1}) \leq 4d_{G'}(u'_s, u'_{s+1})$. Indeed, suppose not: then the path $P'_{u'_s u'_{s+1}}$ satisfies all the conditions of the procedure **CreateNodes** and so u'_s would have been contracted into a vertex with budget C^{r+1}. Since this is not the case, our claim must be true. Adding all these inequalities, we get $d_{G_B}(u, v) \leq 4d_{G'}(u', v') + 2\gamma C^{r+1}$.

Now u' or v' may be a vertex with budget C^{r+1}. Since each edge is of length ϵ, the node w' adjacent to u' on P' is at distance at most ϵ from it. So we can carry out the argument above by replacing u' by w' and v' by a similar node. Theorem A.9 now implies that $d_{G_B}(u, v) \leq 4d_{G'}(u', v') + 2\gamma C^{r+1} + 2\beta C^{r+1} \leq 4d_{G'}(u', v') + \lambda/2C^{r+1}$. ∎

Theorem A.11 *Let G' be the auxiliary graph at the end of stage r (i.e., $G' = G^{r+1}$). If $u', v' \in G'$ with $u = \mathrm{core}(u'), v = \mathrm{core}(v')$, then $d_{G_B}(u, v) \leq 5 \cdot d_{G'}(u', v') + \lambda C^{r+1}$.*

Proof: Let P' be a shortest path between u' and v' in G'. Suppose u'_1 and u'_2 are two vertices with budget C^{r+1} such that $P'_{u'_1 u'_2}$ does not contain any internal node with budget C^{r+1}. Lemma A.10 implies that $d_{G_B}(\mathrm{core}(u_1), \mathrm{core}(u_2)) \leq 4 \cdot d_{G'}(u'_1, u'_2) + \lambda/2 \cdot C^{r+1}$. But we know that $d_{G'}(u'_1, u'_2) \geq 9 \cdot C^{r+2}$ (else the procedure **Contract** would have merged u'_1 and

u_2'), so using this and the the fact that $\lambda < C$ implies $d_{G_B}(\mathrm{core}(u_1), \mathrm{core}(u_2)) \leq 5 d_{G'}(u_1', u_2')$.

Now starting from the left end-point of P', suppose u_1' is the first node with budget C^{r+1} and u_l' is the last node with budget C^{r+1}. Lemma A.10 also implies that the contribution of the paths from left end-point of P' to u_1' and u_l' to right end-point of P' is at most $4 d_{G'}(u', u_1') + 4 d_{G'}(u_l', v') + \lambda C^{r+1}$. By adding the above inequalities for the various portions of the paths, we get the result. ■

A.3 Approximation Ratio

We now need to show that the cost of our solution is within a constant of $\sum_j \alpha_j$. We have made no attempt to optimize the constants here, for clarity of exposition.

Lemma A.12 *If j is alive before the* **Prune Demands** *procedure in stage r, then either $\alpha_j \geq C^{r+1}$ or $j \in D(u')$ for some u' with budget C^{r+1}. If j is alive at the end of stage r, then either $\alpha_j \geq C^{r+1}$ or $j \in D(u')$ for some node u' with budget C^{r+1} with $|D(u')| \geq M$.*

Proof: Consider such a demand j in phase 1 of stage r: if α_j is not raised to C^{r+1} in this phase, this must be because of a frozen facility i' such that $d_{G^r}(i', j) \leq 8 \cdot C^{r+1}$. Now if j is tight with i'' at the end of Phase 2, then Lemma A.2 implies that $d_{G^r}(j, i'') \leq \alpha_j \leq 2C^{r+1}$. The triangle inequality now implies that $d_{G^r}(i', i'') \leq 10 \cdot C^{r+1}$, and hence $i'' \in \mathbf{B}(i')$. In other words, every point that j is tight with lies inside this ball $\mathbf{B}(i')$, and when $\mathbf{B}(i')$ is collapsed into a vertex v' after the **ContractTree** procedure, then $j \in D(v')$. To prove the second part of the lemma, note that if $|D(v')|$ were less than M, we would raise α_j to C^{r+1} in **Regrow**. ■

Lemma A.13 *Let j be alive during the* **Prune Demands** *procedure in stage r. If G' is the auxiliary graph at this time, and j is tight with $i' \in G'$, then $d_{G_B}(j, \mathrm{core}(i'))$ is within a constant of α_j.*

Proof: We prove this by induction on stages. Consider stage 0 : since all the demands are either co-located or at least $20C^2$ apart, each demand raises α_j to C, or else it is co-located with at least $M-1$ other demands, and hence its $\alpha_j = 0$. In both these cases, it is clear that the conditions of the lemma are satisfied.

Now suppose the lemma is true at the end of stage $r-1$. Lemma A.12 implies that at the end of stage $r-1$, either $\alpha_j \geq C^r$ or $\alpha_j \in D(u')$ for some $u' \in F_r$. Let us suppose the former is true, and $\alpha_j \geq C^r$. If j is tight with i', then $d_{G'}(i', j) \leq \alpha_j$ by Lemma A.2. Now applying Theorem A.11 gives us that $d_{G_B}(\mathrm{core}(i), j)$ is at most $O(\alpha_j + C^{r+1})$. But since $\alpha_j \geq C^r$, this in turn is $O(\alpha_j)$. (Note that the constant in the big-Oh depends on C.)

On the other hand, suppose $\alpha_j \in D(u')$ for $u' \in F_r$ at the end of stage $r-1$. By induction, $d_{G_B}(\mathrm{core}(u'), j)$ is within a constant of α_j. u' will be in F_r at the beginning of stage r. In fact u' will get contracted into a node x' with budget C^{r+1} in the procedure $\mathbf{ContractTree}$. Note that $\mathrm{core}(u)$ will be a part of $\mathrm{core}(x)$. So $d_{G_B}(\mathrm{core}(x), j)$ is also within a constant of α_j. Now, $j \in D(x')$ because all the nodes that j was tight with were included in the ball $\mathbf{B}(u')$. If $|D(x')| \geq M$, then x' is the only node that j is tight with. Otherwise, we raise α_j to C^{r+1} and the reasoning in the previous case applies. ■

Theorem A.14 *Suppose s_j, t_j gets removed in the procedure* **Prune Demands** *of stage r. Then $\alpha_{s_j} + \alpha_{t_j}$ can pay for renting edges between them in G_B.*

Proof: We are given that $d_{G'}(s_j, t_j) \leq 5 \cdot C^{r+2}$, and let $s' = s_j(G'), t' = t_j(G')$. Theorem A.11 now implies that $d_{G_B}(\mathrm{core}(s), \mathrm{core}(t)) \leq 5 \cdot 5 \cdot C^{r+2} + \lambda C^{r+1} \leq 26 \cdot C^{r+2}$. By Theorem A.9, $d_{G_B}(s_j, t_j) \leq 26 \cdot C^{r+2} + 2\beta C^{r+1} \leq 27 \cdot C^{r+2}$.

If either of α_{s_j} or α_{t_j} is at least C^{r+1}, then we are done (because the total amount paid would be a $O(\alpha_{s_j} + \alpha_{t_j})$). So suppose that both $\alpha_{s_j}, \alpha_{t_j} < C^{r+1}$. Then Lemma A.12 implies that $s_j \in D(x'), t_j \in D(y')$, where x', y' are nodes with budget C^{r+1}. Furthermore, since $s_j \in D(x')$ implies that $s_j \in G[x']$, by the definition of $D(x')$, and similarly for t_j, $d_{G'}(s_j, t_j) = d_{G'}(x', y')$. But by the procedure **Contract**, we know that $d_{G'}(x', y') \geq 9 \cdot C^{r+2}$. This is possible only if $x' = y'$. Now Lemma A.13 tells us that s_j can pay for a path from s_j to $\mathrm{core}(x')$ and t_j for a path from t_j to $\mathrm{core}(x')$ in G_B, proving the theorem. ■

This shows how the dual variables can pay for the cost of renting paths. Now we need to show how to pay for cost of buying edges. Here *budgets* of the vertices are going to be used, where the budget of a node is roughly the amount it can pay for buying edges. A node can do two things with its budget:

- It can use the budget to buy edges. This can happen only once, after which the node loses its budget. Note that the length of the edges it buys can be within a constant of its budget, since this is a one-time expense.
- It can transfer its budget to some other node. In this case, the amount given must be exactly the same as the amount received, for fear of a cascading of such transfers might end up "creating" unbounded amounts of new wealth incorrectly.

However, wealth *can* be legitimately created — we can create a new node with budget (say) C^{r+1}, and account for its budget in two ways: either (1) we charge it to some α_j's or (2) other nodes transfer their budget to this new node (losing their own budgets in the process, as described in the second bullet above).

Let us now see how this accounting works in detail. In Phase 2, each frozen facility $u' \in F_r$ gets a budget of C^{r+1}. There are M nodes in $P(u')$ and each of them raise their α_j value by C^{r+1}. Now a facility u' can charge its budget to this *increase* in the α_j values of its primary demands. Since different frozen facilities in a fixed stage have disjoint primary demands (by corollary A.7), there is no double counting within a stage. Since we are charging to the *increase* in these dual variables, there is no double counting between two different stages.

Consider the procedure **Contract**: Theorem A.10 implies that the length of P in G_B is at most $O(C^{r+2})$. The node u' pays for the cost of buying edges on P. The budget of v' gets transfered to the budget of the newly created node x'. Note that v' gets contracted into x', so it loses its identity (as well as its budget) from now on. The procedure might also find some nodes w' with a budget of C^{r+1}, in which case the cost of the path joining w and $\mathrm{core}(w')$ is paid by w''s budget. Note that w' will also get merged in x', but there is no transfer of budget .

We next analyze the accounting scheme in procedure **ContractTree**, which is the perhaps the trickiest part in this proof. Let us give some notation first. Suppose procedure **CreateNodes** creates the pair (u', v') in stage r. It finds a shortest path P' between these nodes. Recall that we found a set $\mathcal{D}(u', v') = u'_1, \ldots, u'_\gamma$ of vertices of P'. We need to argue the existence of these points (as promised in the procedure **ContractTree**).

Lemma A.15 *In the procedure* **CreateNodes**, *the set of vertices $\mathcal{D}(u', v')$ of size γ can be found as claimed.*

Proof: Let us use the mapping $\mathcal{P}(u', v', P')$ (described before Lemma A.10) to map P' to a path P joining $\mathrm{core}(u')$ and $\mathrm{core}(v')$. Since $d_{G_B}(\mathrm{core}(v'), \mathrm{core}(u')) \geq \gamma C^{r+1}$, there must be a vertex $x \in P$ such that $d_{G_B}(\mathrm{core}(u'), x) = 7/8 C^{r+1}$ (because all edges are of length ϵ and assume C is a multiple of 8). Let $x' = x(G')$. Since x' is a point in P', and in turn a vertex of G^r, its budget is at most C^r. Now by Theorem A.9, $d_{G_B}(x, \mathrm{core}(x')$ is at most $\beta C^r \ll 1/8 C^{r+1}$. This x' is a candidate for u'_1. One can show similarly that $u'_2, \ldots, u'_\gamma$ exist.

We also need to argue that $u'_1, u'_2, \ldots, u_{\gamma'}$ appear on P' in this order. Let us show u'_2 appears after u'_1. Indeed, $d_{G_B}(\mathrm{core}(u'), \mathrm{core}(v')) \geq \gamma C^{r+1}$, while $d_{G_B}(\mathrm{core}(u'), \mathrm{core}(u'_1)) \leq C^{r+1}$. So there must be a point y on the segment of P joining $\mathrm{core}(u'_1)$ to $\mathrm{core}(v')$ such that $d_{G_B}(\mathrm{core}(u'), x_1$ is $(2 - 1/8) \cdot C^{r+1}$. So if y' corresponds to y in P', then y' is a candidate for u'_2. Thus, we can show that the points $u'_1, u'_2, \ldots, u_{\gamma'}$ appear on P' in this order. ■

For each u'_i we define another point u''_i in P': u''_i is the first point to the right of u'_i on P' such that $d_{G_B}(\mathrm{core}(u'_i), \mathrm{core}(u''_i))$ is at least $1/2C^{r+1}$. Note that $d_{G_B}(\mathrm{core}(u'_i), \mathrm{core}(u'_{i+1})) \geq 3/4 \cdot C^{r+1}$ by the triangle inequality, so u''_i occurs before u'_{i+1} on P'. Let $\mathrm{part}(P')$ denote the union of the segments $P'_{u'_i u''_i}$ for all i. Note that all these segments are disjoint from one another.

Let $\mathrm{zero}(P') = \mathrm{part}(P') \cap Z_r$ be the zero length edges in $\mathrm{part}(P')$. For an edge $e' \in \mathrm{zero}(P')$, let us define a set of demands $\mathrm{Dem}(e')$: e' corresponds to an edge e in the original graph G. $\mathrm{Dem}(e')$ is the set of all those demands which contribute to the left hand side of constraint (3) for e in the dual LP, i.e., those demands j such that $y_{S,j} > 0$ for some S such that $e \in \delta(S)$. Define $\mathrm{Dem}(P') = \cup_{e' \in \mathrm{zero}(P')} \mathrm{Dem}(e')$.

Lemma A.16 *Suppose $j \in \mathrm{Dem}(e')$, where e' corresponds to an edge $e = (u, v)$ in G_B. Suppose j contributes to the left hand side of constraint (3) for e in a previous stage $s \leq r$, and α'_j be the value of its dual variable at that time. Then α'_j is between C^{s+1} and $2C^{s+1}$. Furthermore, $d_{G_B}(j, u)$ and $d_{G_B}(j, v)$ are at most $O(\alpha_j)$.*

Proof: Let e' correspond to an edge $e'' = (u'', v'')$ after phase 2 of stage s. We first claim that $\alpha'_j \geq C^{s+1}$. Indeed, if not, then j would be frozen due to some $i' \in F_r$ and so all nodes that were tight with j, would get included in $\mathbf{B}(i')$. But since the length of e'' is at most ϵ and j is tight with at least one of its endpoints, both these nodes will get included into $\mathbf{B}(i')$. If $s < r$, both u'' and v'' would have been contracted into a single node, and so e' could not be present in the auxiliary graph. On the other hand, if $s = r$, we have found a point in P' which is included in a ball $\mathbf{B}(i')$, which would contradict the construction of P'. This proves that $\alpha'_j \geq C^{s+1}$; the fact that $\alpha'_j \leq 2C^{s+1}$ follows from Lemma A.1.

Now j must have been tight with u'' or v'' in either Phase 1 or 2 of stage s, otherwise it would not have contributed to e'. Hence, by Lemma A.2, $d_{G^s}(j, u'') \leq \alpha'_j$. Now applying Theorem A.11 and using the fact that $\alpha'_j \geq C^{s+1}$ gives us that $d_{G_B}(j, u)$ is within a constant of α'_j. ■

Lemma A.17 *Suppose the pairs (u', v') and (x', y') are created by the procedure* **CreateNodes** *in stage r. Let P' be the path between u' and v' and Q' the path between x' and y'. Then $\mathrm{part}(P')$ and $\mathrm{part}(Q')$ do not share a vertex, and furthermore, $\mathrm{Dem}(P')$ and $\mathrm{Dem}(Q')$ are disjoint.*

Proof: Suppose (u', v') is created before (x', y'). Consider the set $B' = \cup_{u'_i \in \mathcal{D}(u', v')} \mathbf{B}(u'_i)$ — this set B' must contain $\mathrm{part}(P')$ and so Q' can not contain any of these vertices.

Now let $j \in \mathrm{Dem}(e')$ for some edge $e' \in \mathrm{zero}(P')$. Then j must be tight with a vertex in $P'_{u'_i u''_i}$ for some i, which in turn implies that all the nodes that j is tight with must be in the ball $\mathbf{B}(u'_i)$. Thus j cannot be in $\mathrm{Dem}(Q')$. ■

We now have almost all the pieces we need for the proof. The following key theorem accounts for the edges bought by the **ContractTree** procedure.

Lemma A.18 *If a pair (u', v') is created in stage r of* **CreateNode** *procedure, then either*

1. *$\sum_{j \in \mathrm{Dem}(P')} \alpha_j$ is $\Omega(MC^{r+1})$, or*
2. *the sum of the budgets of nodes in $\mathrm{part}(P')$ is at least $2C^{r+1}$.*

Let us assume the lemma is true, and show how this accounts for the edges bought in **ContractTree**, and for the budgets of the newly created nodes. Any edge $e = (u', v')$ in T_X corresponds to buying edges of total length about C^{r+1} (again, using the distance preserving lemmas). If T'_X, a component of T_X, has n' nodes, then we buy edges of total length $O(n' - 1)C^{r+1}$, and we need to account for this. Furthermore, the nodes in T'_X then get contracted into a node with budget C^{r+1}, and we also need to account for this budget.

Suppose all the nodes in T'_X were from F_r created in Phase 1. Since each such node is allotted a budget of C^{r+1}, $n'-1$ of these node budgets can be used to pay for the edges of T'_X, and the remaining one can transfer its budget to the new node created by the contraction.

In not, then T'_X contains a node from $\mathcal{D}(u', v')$. In this case, note that T'_X must contain *all* the nodes in $\mathcal{D}(u', v')$, because the balls created by these nodes overlap sequentially. Since $\mathcal{D}(u', v')$ has $gamma$ nodes, we need to account for at most γ edges in T'_X. Depending on which part of Lemma A.18 holds, we have to look at two cases. If the first case of Lemma A.18 holds, then $\sum_{j \in \mathrm{Dem}(P')} \alpha_j$ can be used to pay for these edges of T_X, and also for the budget C^{r+1} of the newly created node. Note that the amount paid plus the budget may be a constant factor greater than the sum of the α_j's, but since this transfer from dual variables to budgets is a one-time operation, we can get away with it. On the other hand, if we are in second case of the Lemma, then the total budget of $2C^{r+1}$ is divided into two parts: one to pay for the C^{r+1} given as budget for the newly created node, the remaining to account for the γ edges.

We must be careful that we are not double counting. Lemma A.17 shows that no double counting occurs in a single phase. It is possible, of course, that $j \in \mathrm{Dem}(P'), \mathrm{Dem}(Q')$ where P', Q' were created in different phases. But here the geometric scaling comes to the rescue: Lemma A.16 shows that the contribution of α_j in these two cases differs by a factor of at least C, and thus the contribution of a demand to $\mathrm{Dem}(P')$ for all the different paths P' it lies in is a geometric sum which can be bounded by α_j. Thus we will we done if we prove Lemma A.18.

Proof of Lemma A.18: Call a node $u'_i \in \mathcal{D}(u', v')$ *good* if $d_{G'}(u'_i, u''_i) \leq 1/3C^{r+1}$. Note that there are at least $\gamma/4$ nice nodes in $\mathcal{D}(u', v')$. Otherwise, $d_{G'}(u', v')$ will be $> (3\gamma/4) \times (1/3C^{r+1}) = \gamma/4\, C^{r+1}$, a contradiction.

Suppose there is a good u'_i such that the total budget of the nodes in $P'_{u'_i u''_i}$ is less than $8C^{r+1}/\gamma$. If there are no such nodes, then the total budgets of all nodes in $\mathrm{part}(P')$ will be at least $2C^{r+1}$ and we will be done.

Consider the path $P'_{u'_i u''_i}$, and let $\mathrm{Dem}(P'_{u'_i u''_i})$ be the sum of $\mathrm{Dem}(e')$ over all $e' \in P'_{u'_i u''_i}$. We use the mapping $\mathcal{P}(u'_i, u''_i, P'_{u'_i u''_i})$ to map $P'_{u'_i u''_i}$ to a path Q in G_B joining $\mathrm{core}(u'_i)$ and $\mathrm{core}(u''_i)$. We classify the edges in Q into three categories: (1) the edges Q_1 in $P'_{u'_i u''_i}$ which have $\ell(e) = 0$, (2) the edges Q_2 in $P'_{u'_i u''_i}$ which have $\ell(e) = c_e$, and (3) the edges Q_3 which appear in $G_B[x']$ for some some node x' in $P'_{u'_i u''_i}$.

By the choice of u''_i, we have $d_{G_B}(\mathrm{core}(u'_i), \mathrm{core}(u''_i)) \geq 1/2C^{r+1}$. The quantity $\sum_{e \in Q_2} c_e$ is simply $d_{G'}(u'_i, u''_i) \leq 1/3C^{r+1}$. Finally, Theorem A.9 implies that $\sum_{e \in Q_3} c_e$ is at most $2\beta\times$ (total budget of nodes in $P'_{u'_i u''_i}$), which is at most $2\beta \times (8C^{r+1}/\gamma) \leq C^{r+1}/8$. But this implies that

$$d_{G_B}(\mathrm{core}(u'_i), \mathrm{core}(u''_i)) - \textstyle\sum_{e \in Q_2 \cup Q_3} c_e \geq C^{r+1}(\frac{1}{2} - \frac{1}{3} - \frac{1}{8}) = \Omega(C^{r+1} \tag{8}$$

Finally, we construct a graph $H = (V_H, E_H)$, where V_H is the set of all vertices in Q and $\mathrm{Dem}(P'_{u'_i u''_i})$, while E_H contains all the edges in $Q_2 \cup Q_3$. Furthermore, for a vertex $j \in \mathrm{Dem}(P'_{u'_i u''_i})$ and an edge $e = (u, v) \in Q_1$, we join j to u and v if j contributed to the left hand side of constraint (3) for the edge e. Note that Lemma A.16 implies that $d_{G_B}(j, u), d_{G_B}(j, v)$ is $O(\alpha_j)$. Since a node can contribute at most c_e to the left hand side of this constraint, it must be the case that for any edge $e = (u, v) \in Q_1$, there are at least M common neighbors of u and v.

It is now possible to show that there are M paths $R_1, \ldots, R_M$ from $\mathrm{core}(u'_i)$ to $\mathrm{core}(u''_i)$ such that no demand $j \in \mathrm{Dem}(P'_{u'_i u''_i})$ appears in two of these paths. Indeed, assume to the contrary. Now it must be possible to delete $M - 1$ of the demands in $\mathrm{Dem}(P'_{u'_i u''_i})$ and obtain a graph in which $\mathrm{core}(u'_i)$ can reach one end point of some (zero-cost) edge e in Q_1, but not its other endpoint. However, this would mean that all the M demands adjacent to both endpoints of e must have been deleted, which is a contradiction.

But each such R_i is a path of length at most $\sum_{j \in R_i} O(\alpha_j) + \sum_{e \in Q_2 \cup Q_3} c_e$ in G_B. Now substituting this in (8) gives us that $\sum_{j \in R_i} O(\alpha_j)$ is $\Omega(C^{r+1})$. Disjointness of the M paths now completes the proof of the lemma. ■

B Removing Earlier Assumptions

We made some assumptions during the statements of the problems and the description of the algorithm, and here we show how to discharge these assumptions.

1. Assumption from Section 2: all commodities wish to send a single unit of flow. A natural idea to remove this assumption is the following. Given an arbitrary rational amount d_k of flow to send from s_k to t_k, we make $d_k \Delta$ copies of the demand pair (s_k, t_k), where Δ is a sufficiently large integer so that $d_k \Delta$ is integral for each k (M is then replaced by $M\Delta$). As stated, this approach leads to a pseudo-polynomial time algorithm. However, the intuition is correct, and we have to just be careful with the implementation. We ensure that all copies of a single demand behave identically—they have equal α_j values and are tight with the same set of facilities. Hence it suffices to keep track of only one copy of the demand. Let us see how we maintain this property: In Phase 1, we raise the α_j of these demands identically. If one of these demand becomes tight with a facility, all the other demands also become tight with this facility. When a facility counts the number of demands tight with it, it can regard s_k or t_k as contributing $d_k \Delta$ units. The **Regrow** step is implemented in a similar manner. Phase 2 can be trivially modified as well: if we need to raise the α_j value of only $d'_k < d_k \Delta$ of these (imaginary) demands, we raise the dual variables of all the d_k demands at rate $d'_k/(d_k \Delta)$. This will ensure that these $d_k \Delta$ demands always behave the same.

2. Assumption from Subsection 2.1: Facilities can be located at any vertex in the graph. Suppose this is not so, and F is the set of vertices at which the facilities can be located. We construct a new graph $H = (V_H, E_H)$, where $V_H = F$. There is an edge between each pair of vertices in V_H, and the length of this edge is the length of the shortest path between its end-points in G. Each demand is moved to the nearest facility in F. It is easy to show that a solution in this new graph can be mapped to a feasible solution in the original graph with only a factor 2 loss in cost.

3. Assumption from Subsection 3.2: Facilities can be located at any intermediate point of an edge. Suppose $e = (u, v)$ is an edge in the original graph G, and we open a facility at w, a point that lies between u and v in G. w must lie in a connected component; let us call it $C(w)$. Let us partition the demands assigned to w into two parts: A_u being those of the demands assigned to w reaching it via u, and A_v being those reaching it via v, with $|A_u| \leq |A_v|$. If $C(w)$ is singleton, we can move the facility at w to v; reassigning all demands in $A_u \cup A_v$ to v only reduces the cost. If $C(w)$ contains both u and v, then we would not have opened w, so let us say that $v \notin C(w)$. Note that if $C(w)$ contains any point in the portion (w, v), reassigning demands to that point would reduce the connection cost. So $C(w)$ contains a portion $e' = (w', w) \subseteq (u, w)$ of e, where $w' \neq w$. Note that $A_u = \emptyset$, because such demands would prefer w' to w. If $|A_v| \geq M$, we can build edges on the part of e which joins w and v, and reassign all of A_v to v. If $|A_v| < M$, we can avoid building e', and instead directly connect all of A_v to w'. Since all these operations can only reduce the cost of our solution, we can convert our solution to one where facilities lie only on the vertices of the original graph.

4. Assumption from Subsection 3.2: The numbers involved in the problem are small. Note that the algorithm, as defined, is only a pseudo-polynomial time algorithm. This is because subdividing edges may create a large number of vertices. This is easily handled: let e be an edge in E subdivided several times, and let V_e be the new vertices on e. For any demand j, it is easy to see that the set of vertices in V_e that j is tight with is a contiguous segment of e. Maintaining this segment for each demand handles this problem.